GUIA DE ESTUDO • DO ZERO À DISCIPLINA

C do zero às Estruturas de Dados

Uma apostila prática para aprender programação em C, entender memória e dominar as estruturas cobradas na graduação

PARA QUEM É Para estudantes sem experiência prévia que estão cursando Estrutura de Dados. Comece pela Unidade 1 e avance na ordem indicada.

Edição de estudo • 2026

Como usar esta apostila

Você não precisa decorar C inteiro antes de estudar estruturas de dados. Aprenda a linguagem na medida em que ela permite construir, percorrer, testar e liberar estruturas. Digite os exemplos, altere valores e observe o resultado: programação se aprende praticando.

- Estude na ordem: fundamentos, memória e ponteiros, estruturas lineares, árvores, hash e grafos.
- Compile todos os exemplos com avisos ativados e corrija cada aviso.
- Faça os exercícios antes de consultar o gabarito.
- Mantenha um caderno de erros com mensagem, causa e correção.
- Refaça pequenos projetos de memória após alguns dias.

REGRA DE OURO Nunca use um ponteiro sem saber para onde ele aponta, nunca acesse uma posição fora do vetor e libere toda memória alocada dinamicamente.

Trilha sugerida de 10 semanas

Semana	Foco	Entrega
1	Lógica, compilação e tipos	Calculadora
2	Entrada, decisões e laços	Menu interativo
3	Funções, vetores e strings	Cadastro
4	Ponteiros, structs e memória	Vetor dinâmico
5	Complexidade, busca e ordenação	Comparador de algoritmos
6	Recursão e listas	Lista de tarefas
7	Pilhas e filas	Histórico e atendimento
8	Árvores e heaps	Catálogo ordenado
9	Hash e grafos	Índice e rotas
10	Testes e projeto final	Sistema integrado

Mapa do conteúdo

Os títulos usam estilos de cabeçalho do Word. Abra o Painel de Navegação para saltar diretamente entre as unidades.

Bloco	Unidades	Resultado
Base da linguagem	1–10	Escrever programas pequenos com segurança
Memória e modelagem	11–16	Entender ponteiros, alocação, structs e arquivos
Estruturas e algoritmos	17–28	Implementar e analisar estruturas clássicas
Consolidação	29–31	Testar, organizar e preparar-se para provas

Convenções

- Código em fonte monoespaçada deve ser digitado e compilado.
- Caixas verdes trazem boas práticas e amarelas destacam armadilhas.
- Os exemplos usam C17 e evitam extensões específicas.
- Nos algoritmos, n representa a quantidade de elementos da entrada.

UNIDADE 01

O que é programar?

Da ideia ao algoritmo: entrada, processamento e saída

Conceito central

Programar é descrever com precisão uma sequência de ações que o computador consegue executar. Um algoritmo é essa sequência; um programa é o algoritmo escrito em C. Dados são os valores sobre os quais o algoritmo trabalha. Antes do código, identifique a entrada, o processamento e a saída e escreva exemplos concretos.

O que você precisa guardar

- Entenda o enunciado e produza exemplos.
- Divida o problema em partes pequenas.
- Escolha variáveis e estruturas adequadas.
- Implemente e teste casos normais e extremos.

Exemplo para digitar, compilar e modificar

```
ler nota1
ler nota2
media <- (nota1 + nota2) / 2
se media >= 7
    escrever "aprovado"
senao
    escrever "reprovado"
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Descreva entrada, processamento e saída de um programa de troco.
2. Escreva pseudocódigo para achar o maior de três números.
3. Liste três casos de teste para verificar idade mínima.

UNIDADE 02

Ambiente e primeiro programa

Como um arquivo .c vira um executável

Conceito central

Você precisa de editor ou IDE, compilador C e terminal. GCC e Clang são compiladores comuns. O processo passa por pré-processamento, compilação, montagem e ligação. Mensagens de erro devem ser lidas a partir da primeira ocorrência relevante.

O que você precisa guardar

- #include disponibiliza declarações de bibliotecas.
- main é o ponto de entrada.
- Chaves delimitam blocos e instruções costumam terminar em ponto e vírgula.
- Use -Wall -Wextra -Wpedantic e corrija os avisos.

Exemplo para digitar, compilar e modificar

```
#include <stdio.h>

int main(void) {
    printf("Ola, mundo!\n");
    return 0;
}

// gcc -std=c17 -Wall -Wextra -Wpedantic primeiro.c -o primeiro
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Compile e execute o exemplo.
2. Remova um ponto e vírgula, leia o erro e corrija.
3. Altere o texto e inclua uma segunda linha.

UNIDADE 03

Anatomia e organização do código

Cabeçalhos, funções, blocos, comentários e nomes

Conceito central

Um programa é formado por diretivas do pré-processador, declarações, funções e blocos. C diferencia maiúsculas de minúsculas. Prefira nomes que expressem papel, como quantidade_alunos e proximo, e use comentários para explicar intenção ou decisões não óbvias.

O que você precisa guardar

- Protótipo declara a assinatura antes do uso.
- Definição contém o corpo da função.
- // comenta uma linha; /* ... */ cria comentário em bloco.
- Indentação não muda a execução, mas torna a estrutura visível.

Exemplo para digitar, compilar e modificar

```
#include <stdio.h>

int soma(int a, int b);

int main(void) {
    printf("%d\n", soma(2, 3));
    return 0;
}

int soma(int a, int b) { return a + b; }
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Identifique protótipo, chamada e definição.
2. Crie uma função multiplica.
3. Renomeie variáveis vagas para nomes claros.

UNIDADE 04

Variáveis, constantes e tipos

int, float, double, char, bool, size_t e inteiros de largura fixa

Conceito central

Uma variável associa um nome a uma região de memória. O tipo informa como interpretar os bits e quais operações são válidas. Inicialize variáveis na declaração. O tamanho exato de int e long depende da implementação; use sizeof e os limites dos cabeçalhos quando isso importar.

O que você precisa guardar

- int representa inteiros naturais da máquina; signed permite negativos e unsigned faz aritmética modular.
- float tem precisão simples e literais normalmente recebem sufixo f; double oferece maior precisão.
- char guarda um caractere/byte; 'A' é caractere e "A" é string.
- bool vem de <stdbool.h>; size_t é apropriado para tamanhos e índices.
- int32_t e similares vêm de <stdint.h> e expressam largura exata quando disponíveis.

Exemplo para digitar, compilar e modificar

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

int idade = 20;
float temperatura = 36.5f;
double saldo = 1250.75;
char inicial = 'A';
bool ativo = true;
size_t quantidade = 10;
int32_t codigo = 123456;
```

Tipo	Uso	Exemplo	printf
char	caractere/byte	'A'	%c
int	inteiro	42	%d
long long	inteiro amplo	9000000000LL	%lld
float	real simples	3.14f	%f
double	real preciso	3.14159	%f
bool	lógico	true	%d

Tipo	Uso	Exemplo	printf
size_t	tamanho/índice	sizeof(int)	%zu

CUIDADO Dinheiro não deve ser representado ingenuamente com float. Em exercícios, considere centavos como inteiro.

Pratique agora

1. Declare inicial, idade, altura e matrícula com tipos adequados.
2. Imprima sizeof(char), sizeof(int), sizeof(double) e sizeof(void*).
3. Explique a diferença entre 'A' e "A".

UNIDADE 05

Operadores e expressões

Aritmética, comparação, lógica, atribuição e conversões

Conceito central

Operadores aritméticos são +, -, *, / e %. Comparações usam <, <=, >, >=, == e != . Operadores lógicos são &&, || e !. Parênteses tornam a intenção clara. Conversões podem perder informação, e divisão entre inteiros descarta a parte fracionária.

O que você precisa guardar

- % vale para tipos inteiros.
- && e || têm curto-circuito.
- = atribui; == compara.
- Use conversão explícita quando desejar divisão real.

Exemplo para digitar, compilar e modificar

```
int a = 5, b = 2;
double q1 = a / b;          // 2.0
double q2 = (double)a / b;  // 2.5

if (b != 0 && a / b > 1) {
    /* seguro contra divisao por zero */
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Compare 7/2 e 7.0/2.

2. Teste se x está entre 10 e 20 inclusive.

3. Use % para verificar se um número é par.

UNIDADE 06

Entrada e saída

printf, scanf, fgets e validação

Conceito central

printf produz saída formatada; o especificador deve corresponder ao tipo. scanf é útil em exemplos, mas exige endereços, validação do retorno e atenção ao que fica no buffer. Para linhas e entradas robustas, prefira fgets e converta com strtol ou strtod.

O que você precisa guardar

- %d imprime int, %zu size_t, %c char e %s string.
- Em scanf, double usa %lf e float usa %f.
- fgets limita a quantidade e preserva espaços.
- Nunca use gets; valide toda leitura.

Exemplo para digitar, compilar e modificar

```
int idade;
if (scanf("%d", &idade) != 1) {
    fprintf(stderr, "entrada invalida\n");
    return 1;
}
printf("idade: %d\n", idade);
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Leia dois inteiros e imprima a soma.
2. Leia nome completo com fgets.
3. Formate uma média com duas casas.

UNIDADE 07

Decisões

if, else, switch e condições legíveis

Conceito central

Use if/else quando a decisão depende de condições gerais. switch é adequado ao comparar uma expressão inteira ou enum com valores constantes. Sem break, um case continua no próximo; isso deve ser intencional e claro.

O que você precisa guardar

- Organize do caso mais específico para o mais geral.
- Dê nomes a condições complexas usando bool.
- Inclua default quando valores inesperados precisarem de tratamento.
- Evite aninhamento excessivo com retornos antecipados.

Exemplo para digitar, compilar e modificar

```
if (nota >= 7.0) {
    printf("aprovado\n");
} else if (nota >= 5.0) {
    printf("recuperacao\n");
} else {
    printf("reprovado\n");
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Classifique um número como negativo, zero ou positivo.
2. Crie menu de quatro operações.
3. Verifique se um ano é bissexto.

UNIDADE 08

Laços de repetição

while, do-while, for, break e continue

Conceito central

while repete enquanto uma condição for verdadeira e pode executar zero vezes. do-while executa ao menos uma vez. for reúne inicialização, teste e atualização, sendo natural para vetores. Uma invariante descreve o que permanece verdadeiro em cada iteração.

O que você precisa guardar

- Garanta que a condição progrida para o término.
- Em vetor de tamanho n, use i < n.
- break encerra o laço; continue pula para a próxima iteração.
- Laços aninhados frequentemente produzem custo quadrático.

Exemplo para digitar, compilar e modificar

```
int soma = 0;
for (int i = 0; i < 5; i++) {
    soma += i;
}

int opcao;
do {
    scanf("%d", &opcao);
} while (opcao != 0);
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Some os números de 1 a 100.
2. Leia números até zero e calcule média.
3. Imprima uma tabuada com laços aninhados.

UNIDADE 09

Funções e modularização

Parâmetros, retorno, escopo, const e contratos

Conceito central

Uma função possui nome, parâmetros, retorno e comportamento. C passa argumentos por valor. Para modificar uma variável do chamador, passe seu endereço. Variáveis locais existem dentro do bloco; globais criam dependências e devem ser raras.

O que você precisa guardar

- Escreva funções pequenas e coesas.
- Documente pré-condições e efeitos colaterais.
- const comunica que o objeto não será modificado por aquele acesso.
- static local preserva valor entre chamadas; static no arquivo limita visibilidade.

Exemplo para digitar, compilar e modificar

```
double media(const double notas[], size_t n) {
    if (n == 0) return 0.0;
    double soma = 0.0;
    for (size_t i = 0; i < n; i++) soma += notas[i];
    return soma / n;
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Implemente máximo, mínimo e eh_par.
2. Crie troca recebendo dois int*.
3. Separe uma calculadora em funções.

UNIDADE 10

Vetores e matrizes

Coleções contíguas, índices e percursos

Conceito central

Vetores armazenam elementos contiguamente. O acesso por índice é O(1), mas inserir no início exige deslocamentos O(n). Ao passar um vetor para uma função, passe também o tamanho. Matrizes em C são vetores de vetores organizados por linhas.

O que você precisa guardar

- Índices válidos vão de 0 a n-1.
- sizeof v / sizeof v[0] só funciona onde v é realmente vetor.
- Percorra matrizes com laços aninhados.
- A localidade de memória torna vetores eficientes.

Exemplo para digitar, compilar e modificar

```
int notas[5] = {7, 8, 10, 6, 9};
size_t n = sizeof notas / sizeof notas[0];
for (size_t i = 0; i < n; i++) {
    printf("%d\n", notas[i]);
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Encontre maior, menor e média.
2. Inverta um vetor no próprio lugar.
3. Some duas matrizes 3x3.

UNIDADE 11

Strings

Vetores de char, terminador nulo e <string.h>

Conceito central

C não possui um tipo string embutido. Uma string é uma sequência de char terminada por '\0'. O vetor precisa reservar espaço para esse terminador. strlen mede o conteúdo, não a capacidade; strcmp compara strings e == compara endereços.

O que você precisa guardar

- Use fgets para linhas com espaços.
- strcpy e strcat exigem destino suficiente.
- snprintf limita a escrita.
- UTF-8 pode usar vários bytes por caractere; strlen conta bytes.

Exemplo para digitar, compilar e modificar

```
char nome[100];
if (fgets(nome, sizeof nome, stdin)) {
    nome[strcspn(nome, "\n")] = '\0';
    printf("Ola, %s!\n", nome);
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Conte vogais em uma linha.
2. Verifique palíndromo ASCII.
3. Ordene dois nomes com strcmp.

UNIDADE 12

Ponteiros sem mistério

Endereços, desreferenciação, NULL e passagem por endereço

Conceito central

Um ponteiro armazena o endereço de um objeto ou função. & obtém o endereço e * acessa o objeto apontado. O ponteiro não contém o objeto: contém uma localização. Desenhe caixas para objetos e setas para ponteiros.

O que você precisa guardar

- Inicialize ponteiros e use NULL quando não apontarem para objeto.
- Só desreferencie ponteiro válido durante a vida do objeto.
- Após free, não use o endereço.
- Aritmética de ponteiros só é definida dentro do mesmo vetor.

Exemplo para digitar, compilar e modificar

```
int valor = 10;
int *p = &valor;
printf("%p %d\n", (void *)p, *p);
*p = 25;

void troca(int *a, int *b) {
    int t = *a; *a = *b; *b = t;
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Desenhe a memória do exemplo.
2. Use troca para inverter inteiros.
3. Explique por que int *p; *p=3; é inválido.

UNIDADE 13

Memória dinâmica

malloc, calloc, realloc, free e propriedade

Conceito central

Memória dinâmica permite escolher tamanho em execução e manter o bloco até free. Sempre verifique falhas. Defina claramente quem possui o bloco e quem deve liberá-lo. Um ponteiro após free torna-se pendente.

O que você precisa guardar

- malloc aloca sem inicializar; calloc zera bytes.
- Use sizeof *p para acompanhar o tipo.
- Com realloc, guarde o resultado temporariamente.
- Evite vazamento, use-after-free e double free.

Exemplo para digitar, compilar e modificar

```
size_t n = 100;
int *v = malloc(n * sizeof *v);
if (v == NULL) return 1;
for (size_t i = 0; i < n; i++) v[i] = 0;
free(v);
v = NULL;
```

Erro	Significado	Prevenção
vazamento	perda do último ponteiro	defina quem libera
use-after-free	uso após free	encerre o uso e zere
double free	liberação repetida	uma responsabilidade
overflow	tamanho de alocação estoura	valide antes de multiplicar

Pratique agora

1. Aloque n doubles e calcule média.
2. Implemente crescimento por duplicação.
3. Localize um vazamento em código de teste.

UNIDADE 14

Struct, enum, union e typedef

Modelagem de registros e estados

Conceito central

Struct agrupa campos relacionados e fundamenta nós de listas e árvores. O operador ponto acessa membro de objeto; -> acessa membro por ponteiro. Enum cria nomes para constantes inteiras. Union sobrepõe membros na mesma memória. typedef cria um apelido de tipo.

O que você precisa guardar

- O compilador pode inserir padding para alinhamento.
- Use enum com switch para estados fechados.
- Em union, interprete apenas o membro ativo.
- Structs podem conter ponteiros para estruturas do mesmo tipo.

Exemplo para digitar, compilar e modificar

```
typedef enum { PENDENTE, ANDAMENTO, CONCLUIDA } Estado;
typedef struct {
    int matricula;
    char nome[60];
    double media;
    Estado estado;
} Aluno;

Aluno a = {2026001, "Lia", 8.5, PENDENTE};
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Modele Produto com código, nome, preço e estoque.
2. Use enum de prioridade em switch.
3. Ordene Aluno por média.

UNIDADE 15

Arquivos e persistência

Texto, binário, abertura, leitura, escrita e fechamento

Conceito central

FILE representa um fluxo. fopen abre com um modo; sempre verifique NULL. Texto é legível e portátil, mas exige conversão. Gravar uma struct bruta em binário não garante portabilidade por causa de padding, tamanhos e endianness.

O que você precisa guardar

- r lê, w trunca e escreve, a acrescenta no final.
- Use perror para diagnóstico baseado em errno.
- Verifique retornos de leitura e escrita.
- Feche todo arquivo e trate falha de fclose.

Exemplo para digitar, compilar e modificar

```
FILE *arq = fopen("dados.txt", "w");
if (arq == NULL) { perror("dados.txt"); return 1; }
fprintf(arq, "%d;%s;%.2f\n", 1, "Teclado", 99.90);
if (fclose(arq) != 0) perror("fclose");
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Salve dez números e leia-os.
2. Conte linhas de um arquivo.
3. Exporte produtos como CSV.

UNIDADE 16

Projetos com vários arquivos

Cabeçalhos, implementação, include guards e build

Conceito central

Cabeçalhos contêm interfaces: tipos, constantes e declarações. Arquivos .c contêm implementações. Include guards evitam inclusão repetida. Compile módulos em objetos e depois ligue-os. Macros são substituição textual e exigem cautela.

O que você precisa guardar

- Não coloque definições globais comuns em cabeçalhos.
- Inclua no .c o próprio cabeçalho do módulo.
- Mantenha detalhes privados como static no .c.
- Automatize o build com Make, CMake ou ferramenta da disciplina.

Exemplo para digitar, compilar e modificar

```
// vetor.h
#ifndef VETOR_H
#define VETOR_H
#include <stddef.h>
int maximo(const int v[], size_t n);
#endif

// build: gcc -std=c17 -Wall -Wextra vetor.c main.c -o app
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Separe calculadora.h, calculadora.c e main.c.
2. Crie um Makefile simples.
3. Explique o perigo de #define QUADRADO(x) x*x.

UNIDADE 17

Erros e depuração

Avisos, depurador, sanitizers e comportamento indefinido

Conceito central

Erros podem ocorrer na sintaxe, ligação, lógica ou memória. Reproduza com entrada mínima, leia a primeira mensagem relevante, formule uma hipótese e inspecione valores. Comportamento indefinido significa que C não promete resultado; parecer funcionar não prova correção.

O que você precisa guardar

- Compile com -g para depurar.
- AddressSanitizer detecta várias violações de memória.
- UndefinedBehaviorSanitizer detecta certas operações indefinidas.
- Teste novamente após corrigir a causa, não apenas o sintoma.

Exemplo para digitar, compilar e modificar

```
gcc -std=c17 -Wall -Wextra -Wpedantic -g \
    -fsanitize=address,undefined programa.c -o programa
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Acompanhe um laço no depurador.
2. Provoque v[5] em int v[5] com sanitizer.
3. Liste cinco causas de comportamento indefinido.

UNIDADE 18

TADs e invariantes

A ponte entre C e Estrutura de Dados

Conceito central

Um tipo abstrato de dados descreve valores e operações observáveis, independentemente da representação. Uma pilha oferece push, pop, topo e vazio, seja implementada com vetor ou lista. Invariantes registram propriedades que toda operação válida deve preservar.

O que você precisa guardar

- Interface diz o que pode ser feito.
- Representação diz como os dados ficam guardados.
- Invariante define estados válidos.
- Complexidade mede o custo das operações.

Exemplo para digitar, compilar e modificar

```
typedef struct Pilha Pilha;
Pilha *pilha_criar(void);
bool pilha_push(Pilha *p, int valor);
bool pilha_pop(Pilha *p, int *saida);
bool pilha_vazia(const Pilha *p);
void pilha_destruir(Pilha *p);
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Escreva interface de um conjunto de inteiros.
2. Defina invariantes de fila circular.
3. Explique como esconder a struct protege a abstração.

UNIDADE 19

Complexidade

Big O de tempo e espaço

Conceito central

Big O descreve como o custo cresce quando n aumenta. Ignoramos constantes e termos menores para comparar escalabilidade. Analise melhor, médio ou pior caso conforme o contexto. Memória auxiliar e pilha de chamadas também contam no espaço.

O que você precisa guardar

- O(1): acesso por índice.
- O(log n): busca binária.
- O(n): percurso completo.
- O(n log n): ordenações eficientes.
- O(n^2): dois laços completos aninhados.

Exemplo para digitar, compilar e modificar

```
// O(n)
for (size_t i = 0; i < n; i++) soma += v[i];

// O(n^2)
for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < n; j++) usa(v[i], v[j]);
```

Classe	Exemplo	Ao dobrar n
O(1)	v[i]	quase igual
O(log n)	busca binária	mais uma etapa
O(n)	percorrer vetor	aprox. dobra
O(n log n)	merge sort	pouco mais que dobra
O(n^2)	pares	aprox. quadruplica

Pratique agora

1. Classifique laços de exercícios da disciplina.
2. Explique por que 5n+20 é O(n).
3. Compare acesso, busca e inserção em vetor e lista.

UNIDADE 20

Recursão

Caso base, passo recursivo e pilha de chamadas

Conceito central

Toda recursão correta precisa atingir um caso base. Cada chamada ocupa um quadro com parâmetros e variáveis locais; profundidade excessiva pode estourar a pilha. Recursão combina naturalmente com árvores, divisão e conquista e backtracking.

O que você precisa guardar

- O subproblema deve ficar estritamente menor.
- Conte tempo e profundidade da pilha.
- Fibonacci ingênuo repete subproblemas.
- Prefira iteração quando for mais simples e segura.

Exemplo para digitar, compilar e modificar

```
unsigned long long fatorial(unsigned n) {
    if (n <= 1) return 1;
    return n * fatorial(n - 1);
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Implemente soma recursiva de vetor.
2. Inverta string recursivamente.
3. Desenhe a pilha de fatorial(4).

UNIDADE 21

Busca e ordenação

Busca linear/binária e algoritmos de ordenação

Conceito central

Busca linear funciona em qualquer vetor e custa O(n). Busca binária reduz pela metade um intervalo ordenado e custa O(log n). Ao ordenar, considere estabilidade, memória extra e pior caso.

O que você precisa guardar

- Busca binária exige a mesma ordem usada na comparação.
- Insertion sort é bom para poucos dados quase ordenados.
- Merge sort é O(n log n); quicksort é rápido em média, mas pode chegar a $O(n^2)$.
- qsort exige um comparador correto.

Exemplo para digitar, compilar e modificar

```
bool busca_binaria(const int v[], size_t n, int alvo) {
    size_t ini=0, fim=n;
    while (ini < fim) {
        size_t m=ini+(fim-ini)/2;
        if (v[m]==alvo) return true;
        if (v[m]<alvo) ini=m+1; else fim=m;
    }
    return false;
}
```

Algoritmo	Tempo	Extra	Estável
Selection	$O(n^2)$	O(1)	não
Insertion	$O(n^2)$	O(1)	sim
Merge	O(n log n)	O(n)	sim
Quicksort	médio O(n log n)	pilha	não garantido

Pratique agora

1. Implemente selection sort.
2. Teste duplicatas e vetor reverso.
3. Crie comparador de Aluno para qsort.

UNIDADE 22

Listas encadeadas

Nós, inserção, remoção, percursos e liberação

Conceito central

Uma lista liga nós por ponteiros. Acesso por índice e busca custam O(n), mas inserir no início custa O(1). Nas remoções, preserve o próximo nó antes de liberar e religue ponteiros sem perder o restante. Desenhar setas evita muitos erros.

O que você precisa guardar

- O início é NULL quando a lista está vazia.
- No** permite alterar o ponteiro inicial do chamador.
- Lista dupla facilita movimentos nas duas direções.
- Libere todos os nós e zere o início.

Exemplo para digitar, compilar e modificar

```
typedef struct No { int valor; struct No *prox; } No;

bool inserir_inicio(No **inicio, int valor) {
    No *novo = malloc(sizeof *novo);
    if (!novo) return false;
    novo->valor=valor; novo->prox=*inicio; *inicio=novo;
    return true;
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Implemente imprimir, buscar e contar.
2. Remova a primeira ocorrência.
3. Destrua a lista sem vazamentos.

UNIDADE 23

Pilhas, filas e deques

LIFO, FIFO, fila circular e aplicações

Conceito central

Pilha segue LIFO e serve para desfazer, expressões e DFS. Fila segue FIFO e serve para atendimento e BFS. Deque permite inserir e remover nas duas extremidades. Implementações com vetor circular evitam deslocar elementos.

O que você precisa guardar

- push/pop e enqueue/dequeue devem ser O(1), salvo realocação.
- Não use -1 como erro se -1 puder ser dado válido.
- Fila circular mantém início, tamanho e capacidade.
- Índice de inserção: (inicio+tamanho)%capacidade.

Exemplo para digitar, compilar e modificar

```
bool fila_remover(Fila *f, int *saida) {
    if (f->tamanho == 0) return false;
    *saida = f->dados[f->inicio];
    f->inicio = (f->inicio + 1) % f->capacidade;
    f->tamanho--;
    return true;
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Implemente pilha dinâmica.
2. Verifique parênteses balanceados.
3. Simule fila circular de atendimento.

UNIDADE 24

Árvores e BST

Hierarquia, travessias e ordenação por chave

Conceito central

Árvores modelam hierarquia. Uma árvore binária tem até dois filhos por nó. Em uma BST, chaves menores ficam à esquerda e maiores à direita conforme a política de duplicatas. Sem balanceamento, ela pode degenerar em lista e custar O(n).

O que você precisa guardar

- Pré-ordem: nó, esquerda, direita.
- Em ordem: esquerda, nó, direita; produz ordem crescente na BST.
- Pós-ordem: esquerda, direita, nó; útil para liberar.
- Por níveis usa fila.

Exemplo para digitar, compilar e modificar

```
typedef struct NoA {
    int chave;
    struct NoA *esq, *dir;
} NoA;

NoA *buscar(NoA *r, int x) {
    while (r && r->chave != x) r = x < r->chave ? r->esq : r->dir;
    return r;
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Implemente inserção em BST.
2. Imprima em ordem.
3. Destrua em pós-ordem.

UNIDADE 25

Heaps e prioridade

Acesso eficiente ao mínimo ou máximo

Conceito central

Um min-heap mantém cada pai menor ou igual aos filhos. A árvore completa cabe em vetor: pai=(i-1)/2, esquerdo=2i+1 e direito=2i+2. Consultar mínimo custa O(1); inserir e remover custam O(log n). Construir por heapify custa O(n).

O que você precisa guardar

- Inserir adiciona ao fim e sobe.
- Remover troca a raiz pelo último e desce.
- Heap não mantém todos os itens em ordem global.
- Fila de prioridade é um TAD; heap é uma implementação comum.

Exemplo para digitar, compilar e modificar

```
void descer(int v[], size_t n, size_t i) {
    for (;;) {
        size_t m=i, e=2*i+1, d=2*i+2;
        if (e<n && v[e]<v[m]) m=e;
        if (d<n && v[d]<v[m]) m=d;
        if (m==i) break;
        int t=v[i]; v[i]=v[m]; v[m]=t; i=m;
    }
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Desenhe o heap [2,5,3,9,7,8].
2. Implemente inserção.
3. Compare fila comum e prioridade.

UNIDADE 26

Tabelas hash

Chaves, colisões e fator de carga

Conceito central

Uma tabela hash transforma chave em índice de balde. Busca, inserção e remoção têm O(1) médio sob hipóteses adequadas, mas O(n) no pior caso. Colisões são inevitáveis e podem ser tratadas por encadeamento ou endereçamento aberto.

O que você precisa guardar

- Hash igual não implica chave igual; confirme igualdade.
- Chaves iguais precisam produzir hashes iguais.
- Fator de carga é elementos/baldes.
- Ao redimensionar, recalcule os índices de todos os itens.

Exemplo para digitar, compilar e modificar

```
unsigned long hash_texto(const char *s) {
    unsigned long h = 5381;
    while (*s) h = h * 33u + (unsigned char)*s++;
    return h;
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Implemente hash de inteiros com encadeamento.
2. Conte colisões.
3. Explique por que redimensionar exige rehash.

UNIDADE 27

Grafos

Vértices, arestas, representações, BFS e DFS

Conceito central

Grafos modelam relações como cidades e estradas, pessoas e amizades ou disciplinas e pré-requisitos. Podem ser direcionados ou não e ponderados ou não. Matriz de adjacência usa $O(V^2)$; lista usa $O(V+E)$ e é boa para grafos esparsos.

O que você precisa guardar

- BFS usa fila e visita por níveis.
- DFS usa pilha ou recursão.
- Marque visitados para evitar ciclos.
- Com lista, BFS e DFS custam $O(V+E)$.

Exemplo para digitar, compilar e modificar

```
BFS(origem):
    marcar origem e enfileirar
    enquanto fila nao vazia:
        u <- desenfileirar
        para cada vizinho v de u:
            se v nao marcado:
                marcar v e enfileirar
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Modele cinco cidades em lista de adjacência.
2. Execute BFS manualmente.
3. Explique matriz versus lista.

UNIDADE 28

Escolhendo a estrutura

Decisão baseada nas operações dominantes

Conceito central

Não existe uma estrutura campeã. Escolha segundo ordem necessária, operação dominante, tamanho, acesso por índice, limites de memória e propriedade dos dados. Complexidade é parte da decisão, junto com simplicidade, localidade e confiabilidade.

O que você precisa guardar

- Vetor: índice O(1).
- Lista/deque: mudanças nas extremidades.
- Pilha: LIFO; fila: FIFO.
- Heap: prioridade; hash: chave-valor médio rápido.
- Árvore ordenada: busca e percurso em ordem; grafo: relações gerais.

Necessidade	Estrutura	Motivo
índice rápido	vetor	O(1)
início frequente	lista/deque	sem deslocar
último primeiro	pilha	LIFO
primeiro primeiro	fila	FIFO
prioridade	heap	topo rápido
chave-valor	hash	O(1) médio
ordem por chave	árvore balanceada	O(log n)
relações	grafo	conexões

Pratique agora

1. Escolha estruturas para histórico, pronto-socorro e dicionário.
2. Justifique vetor versus lista em catálogo estável.
3. Faça matriz de operações do seu projeto.

UNIDADE 29

Testes e qualidade

Casos extremos, invariantes e ferramentas

Conceito central

Teste estrutura vazia, um elemento, capacidade cheia, crescimento, valores duplicados e ausentes. assert é útil para invariantes e testes internos. Um programa de qualidade compila sem avisos, trata falhas e libera recursos.

O que você precisa guardar

- Teste primeiro, último e ausente.
- Verifique invariantes após sequências longas.
- Execute sanitizers e analise vazamentos.
- Mantenha funções pequenas, const correto e build reprodutível.

Exemplo para digitar, compilar e modificar

```
#include <assert.h>
void testar_busca(void) {
    int v[] = {2,4,6,8};
    assert(busca_linear(v,4,2) == 0);
    assert(busca_linear(v,4,8) == 3);
    assert(busca_linear(v,4,5) == -1);
}
```

BOA PRÁTICA Digite o exemplo, compile com avisos e altere pelo menos uma parte antes de avançar.

Pratique agora

1. Teste pilha vazia e unitária.
2. Teste ordenação com duplicatas.
3. Escreva verifica_invariante para fila.

UNIDADE 30

Projeto final guiado

Gerenciador de tarefas integrando o conteúdo

Conceito central

Construa um sistema de terminal. Cada tarefa tem id, título, prioridade e estado. Use vetor dinâmico ou lista como repositório, fila ou heap para execução, pilha para desfazer e arquivo texto para persistência. Desenvolva em etapas pequenas e testadas.

O que você precisa guardar

- Modele Tarefa e enums.
- Implemente cadastro, busca e atualização.
- Separe módulos .h/.c.
- Adicione ordenação, fila e histórico.
- Salve, carregue, teste e execute sanitizers.

Módulo	Responsabilidade	Estrutura
tarefa	modelo e validação	struct/enum
repositorio	armazenamento	vetor/lista
agenda	ordem de execução	fila/heap
historico	desfazer	pilha
persistencia	ler e salvar	FILE
app	menu e coordenação	funções

CRITÉRIO DE PRONTO Compila sem avisos, passa nos testes, não acusa sanitizers, trata arquivo inexistente e libera toda memória.

Pratique agora

1. Escreva tarefa.h.
2. Defina códigos de erro.
3. Desenhe o fluxo do menu à persistência.

UNIDADE 31

Revisão, gabarito e próximos passos

O que saber para prova e como continuar

Conceito central

Para a prova, explique valor, endereço e ponteiro; trace estruturas após operações; compare custos; reconheça vazamentos; simule BFS, DFS, heap e fila circular; e justifique pré-condições. Mostre estados intermediários: eles tornam seu raciocínio verificável.

O que você precisa guardar

- Par: n % 2 == 0; intervalo: x>=10 && x<=20.
- int *p; *p=3 é inválido porque p é indeterminado.
- Dois laços completos aninhados produzem O(n²).
- Busca binária exige ordenação.
- Ao mudar capacidade de hash, todos os índices mudam.

Pergunta	Resposta curta
Por que & em scanf?	Fornece o endereço onde escrever.
free apaga o ponteiro?	Não; o endereço armazenado fica inválido.
Vetor ou lista para índice?	Vetor: O(1); lista: O(n).
Hash é sempre O(1)?	Não; O(1) médio, O(n) pior.
Por que BST pode ser O(n)?	Pode degenerar sem balanceamento.

NA PROVA Mostre o estado da estrutura após cada operação; o rastreamento claro revela o raciocínio.

Pratique agora

1. Refaça uma lista, pilha e fila sem consultar.
2. Crie folha própria de complexidades.
3. Trace BST da sequência 8,3,10,1,6,14.

Folha de consulta rápida

Objetivo	Forma em C
inteiro	int x = 0;
real	double x = 0.0;
caractere	char c = 'A';
texto	char s[50] = "texto";
endereço	&x
objeto apontado	*p
membro	obj.campo / ptr->campo
alocar	T *p = malloc(n * sizeof *p);
liberar	free(p); p = NULL;
percorrer	for (size_t i=0; i<n; i++)
imprimir size_t	printf("%zu", n);
testar ponteiro	if (p == NULL)

Comandos de compilação

```
gcc -std=c17 -Wall -Wextra -Wpedantic *.c -o programa

gcc -std=c17 -Wall -Wextra -Wpedantic -g \
    -fsanitize=address,undefined *.c -o programa

./programa
```

ANTES DE ENTREGAR Compile sem avisos, teste extremos, execute sanitizers, verifique retornos e confirme que cada malloc tem um free correspondente.